COS 424: Interacting with Data

Lecturer: Robert Schapire
Scribe: Megan Lee

Lecture # 5
February 20, 2007

1 Decision Trees

During the last class, we talked about growing decision trees from a dataset. In doing this, there are two conflicting goals:

- achieving a low training error
- building a tree that is not too large

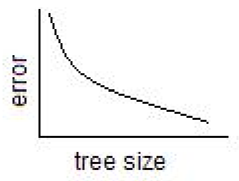
We discussed a greedy, heuristic algorithm that would try to achieve both of these conflicting goals.

2 Classification Error

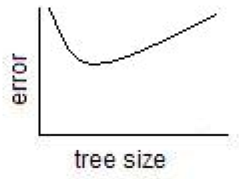
Suppose that we cut off the growing process at various points over the growing processs, and we evaluate the error of the tree at that point and time. This would lead to a graph of size vs. error (where error is the probability of making a mistake). There are two error rates to be considered:

- training error (i.e. fraction of mistakes made on the training set)
- testing error (i.e. fraction of mistakes made on the testing set)

The error curves are as follows:



tree size vs. training error



tree size vs. testing error

As the tree size increases, training error decreases. However, as the tree size increases, testing error decreases at first since we expect the test data to be similar to the training data, but at a certain point, the training algorithm starts training to the noise in the data, becoming less accurate on the testing data. At this point we are no longer fitting the data and instead fitting the noise in the data. Therefore, the shape of the testing error curve will start to increase at a certain point at which the tree is too big and too complex to perform well on testing data (an application of Occam's razor). This is called overfitting to the data, in which the tree is fitted to spurious data. As the tree grows in size, it will fit the training data perfectly and not be of practical use for other data such as the testing set. We want to choose a tree at the minimum of the curve, but we are not aware of the test curve during training. We build the tree only using the training error curve, which appears to be decreasing with tree size. Again, we have two conflicting goals. There is a tradeoff between training error and tree size.

3 Tree Size

There are two general methods of controlling the size of the tree:

- grow the tree more carefully and try to end the growing process at an appropriate point early on
- grow the biggest tree possible (one that completely fits the data), then prune it to be smaller (this is the more common method)

One common technique is to separate the training set into two parts, the growing set and the pruning set. The tree is grown using only the growing set, and the pruning set is used to estimate the testing error of all possible subtrees that can be built, and the subtree with the lowest error on the pruning set is chosen as the decision tree. In this method, we are using the pruning set as a proxy for the testing set with the hope of achieving a curve similar to the test curve when using the pruning set. For an example, 2/3 of the training set may be used for growing, while 1/3 is used for pruning. A disadvantage of this method is that training data is wasted, a serious problem if the dataset is small. Another approach is to try to explicitly optimize a tradeoff between the number of errors and the size of the tree. Consider the value

$$\#\text{training errors} + \textit{constant} \times \text{size of tree}$$

Now there is only one value that must be minimized to determine the optimal tree. This value attempts to capture the two conflicting interests simultaneously.

4 Assumptions in creating decision trees

As with any algorithm, there are various assumptions that are made when building decision trees. Three of these assumptions are that:

- The data can be described by features, such as the features of Batman characters. Sometimes we assume these features are discrete, but we can also use decision trees when the features are continuous. Binary decisions are made on the basis of continuous features by determining a threshold that divides the range of values into intervals correlated with decisions.

- The class label can be predicted using a logical set of decisions that can be summarized by the decision tree.
- The greedy procedure will be effective on the data that we are given, where effectiveness is achieved by finding a small tree with low error.

5 Decision tree history

Decision trees have been widely used since the 1980s. CART was an algorithm widely used in the statistical community, and ID3 and its successor, C4.5, were dominant in the machine learning community. These algorithms are fast procedures, fairly easy to program, and interpretable (i.e. understandable). A drawback of decision trees is that they are generally not as accurate as other machine learning methods. We will be looking at some of these state of the art algorithms later in the course.

It is difficult to explain why decision trees are not optimally accurate. Decision trees may fail if the data is probabilistic or if the data is noisy. Features in the tree are not weighted and simplicity is hard to control, with overfitting a constant problem while growing the tree.

6 Theory: a mathematical model for the learning problem

Until now, we've taken a very intuitive approach. Although we know that we need data, low error, and a simple rule, there are still many unresolved questions. For an example, what does it mean for a rule to be simple? Why is simplicity so important? How much data is enough? What can we guarantee about accuracy? And how can we explain overfitting? We want to formalize the learning problem and define a measure of complexity.

6.1 Data

Training and testing examples should be similar. For an example, if we were classifying images of handwritten digits by the digits they represent, we would want training examples of the digits 0-9, not, for example, only 0-5. In the latter case, the algorithm would fail miserably. Generally, the testing and training examples can be similar if they are produced by the same process.

The following is a formalization of this idea of the testing and training examples being generated by the same process. Assume that the data is random, and the testing and training examples are generated by the same source, a distribution D. From this distribution D, we get an example, x. The distribution is unknown, but all examples are IID since they originate from the same distribution. There is also a target function, $c(x)$, that indicates the true label of each example. During training, the learning algorithm is given a set of examples, $x_1, \ldots, x_n$, each from the distribution D, and each example is labeled with its correct label, $c(x_1), \ldots, c(x_n)$. This data is fed to the learning algorithm, which outputs the prediction rule (i.e. hypothesis), which can take in a new example, x, and output a prediction $h(x)$. We measure the goodness of the classifier by determining the generalization error, which is the probability that a new example x, chosen at random with respect to the distribution D, will be misclassified. This is equivalent to the expected test error, which can also be denoted as $err(h)$. Our goal is to minimize the generalization error.

$$\begin{aligned}\text{generalization error} &= Pr_{x\in D}[h(x) \neq c(x)] \\ &= E[\text{test error}] \\ &= err(h)\end{aligned}$$

6.2 Complexity

Complexity here is an informal term defined as the opposite, or lack of, simplicity. Consider the two trees, h_A, which consists of five nodes, and h_B, which consists of one node. Obviously, h_A is more complex. But why is this so? One way of thinking about the complexities of these trees is to consider the length of the programs needed to compute the trees. That is, how many bits or ascii characters would we need to write the tree down? The program for h_A would be far longer than the program for h_B, which would be one line. But this is a slippery idea. Suppose that in the next edition of Java, there was a library function that exactly computed tree h_A. Now, h_A only requires one line of code, and our definition of complexity is invalid. Although description length as complexity is an intuitive idea, it is a rather slippery measure. That is, complexity should not be measured for a single rule.

Instead, it is actually a *class* of rules that is simple or complex. In this case, h_A belongs to a class of hypotheses, H_5, which is the set of all trees with five nodes. The real reason why h_A is more complex than h_B is because h_A belongs to class H_5, which is more complex, or richer, than class H_1, which is the set of trees with one node. Now, why is H_5 richer than H_1? H_5 must be bigger because it contains all trees in H_1. H_5 is the set of all possible trees that have five nodes. For now, we are assuming that trees in the same class have the same complexity. This prevents us from thinking about the complexity of individual trees rather than the complexity of a set of trees. Our claim is that we need more data to learn with the same accuracy in a large hypothesis space than in a small hypothesis space. Essentially, the size of the hypothesis space is important.

$$|H_5| > |H_1|$$

To illustrate this point, imagine this class experiment. There are training objects 1-10 and testing objects 11-20. Each person in the class represents a hypothesis, and each person writes down his/her hypothesis (0 or 1) for each object in the training set. When the true labels of the training set are revealed, each person calculates his error. During class, the most accurate hypothesis was 80%. The two people who achieved 80% accuracy then checked their accuracy on the testing set, on which they were 30-40% accurate. However, everybody's true generalization error is 50%, since the target values were generated by randomly flipping a coin. Regardless, since there are many people and many hypotheses, there is a chance that a few people will have high accuracy during training. When there is a large set of hypotheses, there is a good chance that one of them might seem to do well on the training set, although it may still perform poorly on a testing set. In this way, the testing accuracy somehow depends on the size of the hypothesis space. (The hypothesis space exists before any data is seen. Once we learn the data, the best hypothesis in this space is determined.)

6.3 Theorem

Let $\lg|H|$ be a measure of the complexity (i.e. size) of the hypothesis space in terms of the number of bits required to describe any hypothesis in the space. This complexity is proportional to the size of the tree.

$$complexity = lg\,|H| = O(n)$$

where n is the number of nodes in the tree.

Theorem 1 *Say that algorithm* A *finds a hypothesis* $h_A \in$ H*. This hypothesis is consistent (no mistakes on training set) with all* m *training examples. That is, assume that the training error is zero. Then,*

$$err(h_A) \leq \frac{ln|H| + ln\frac{1}{\delta}}{m}$$

with probability greater than $1-\delta$ *(i.e., with high probability).*

δ, a small constant, is included because the algorithm cannot be perfect. Whenever we are using random data, there is the possibility of getting a weird dataset. For an example, the algorithm would perform badly if the number 7 never appeared in a set of zip codes. In this way, there is always a small chance that an algorithm will perform very poorly.

This relation says that as the amount of data, m, increases, the error decreases. Also, as the complexity increases, the error increases as well. The proof follows.

Proof.

$$h_A \in H$$

We want to show that

$$err(h_A) \leq \frac{ln|H| + ln\frac{1}{\delta}}{m}$$

Let

$$\epsilon = \frac{ln|H| + ln\frac{1}{\delta}}{m}$$

The hypothesis h is ϵ-bad if *err(h)* $> \epsilon$. Show that h_A is not ϵ-bad with probability $\geq 1-\delta$. That is,

$$Pr[h_A \text{ is not } \epsilon\text{-bad}] \geq 1-\delta$$

So prove:

$$Pr[h_A \text{ is } \epsilon\text{-bad}] \leq \delta$$

We are proving a bound on the probability. h_A is consistent with the data, so

$$Pr[h_A \text{ is } \epsilon\text{-bad}] = \text{Pr}[h_A \text{ is consistent and } \epsilon\text{-bad}]$$

If the condition of this probability holds, then there is some hypothesis that is consistent and ϵ-bad. Thus,

$$Pr[h_A \text{ is consistent and } \epsilon\text{-bad}] \leq Pr[\exists h \in H : h \text{ is consistent and } \epsilon\text{-bad}] \tag{1}$$

Let

$$\begin{aligned} B &= \{h \in H : h \text{ is } \epsilon\text{-bad}\} \\ &= \{h_1, \ldots, h_k\} \end{aligned}$$

Then (1) is equal to

$$\begin{aligned} & Pr[\exists h \in B : h \text{ is consistent}] \\ = \; & Pr[h_1 \text{ is consistent} \vee \ldots \vee h_k \text{ is consistent}] \end{aligned}$$

This is the probability that any h is bad.

Use the union bound,

$$Pr[a \vee b] \leq Pr[a] + Pr[b]$$

so

$$Pr[h_1 \text{ is consistent} \vee \ldots \vee h_k \text{ is consistent}] \leq Pr[h_1 \text{ is consistent}] + \ldots + Pr[h_k \text{ is consistent}]$$

Now compute the probability that any $h \in$ B is consistent with the training set.

$$Pr[h \text{ is consistent}] = Pr[h(x_1) = c(x_1) \wedge \ldots \wedge h(x_m) = c(x_m)]$$

This becomes a product because the examples are IID.

$$= Pr[h(x_1) = c(x_1)] \times \ldots \times Pr[h(x_m) = c(x_m)]$$

For each x,

$$Pr[h(x) = c(x)] = 1 - err(h) \leq 1 - \epsilon$$
$$Pr[h \text{ is consistent}] \leq (1 - \epsilon)^m$$

$$\begin{aligned} Pr[h_A \text{ is consistent and } \epsilon\text{-bad}] &\leq Pr[h_1 \text{ is consistent}] + \ldots + Pr[h_k \text{ is consistent}] \\ &\leq |B|(1-\epsilon)^m \\ &\leq |H|(1-\epsilon)^m \\ &\leq |H|e^{-\epsilon m} \\ &= \delta \end{aligned}$$

since

$$1 + x \leq e^x \forall x$$

and since the definition of ϵ is

$$\epsilon = \frac{ln|H| + ln\frac{1}{\delta}}{m}$$

We have shown that Pr[h_A is ϵ-bad] $\leq \delta$